RECEIVED
CENTRAL FAX CENTER

MAR 14 2006

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellants:	Steven O. HOBBS et al.	§	Confirmation No.:	4992
Serial No.:	09/785,143	§	Group Art Unit:	2192
Filed:	02/16/2001	§	Examiner:	Andre R. Fowlkes
For:	Method And Apparatus For Reducing Cache Thrashing	§	Docket No.:	200301732-1

I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571) 273-8300.
Date of facsimile: 03/14/2006
Typed Name: Colleen F. Brown
Signature: Colleen F. Brown

RESPONSE TO NOTIFICATION OF NON-COMPLIANT APPEAL BRIEF (37 CFR 41.37)

Mail Stop Appeal Brief – Patents
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

March 14, 2006

Sir:

In response to the Notification of Non-Compliant Appeal Brief dated February 17, 2006, Appellants submit a complete new Appeal Brief. In said Notice, the Examiner objected to Appellants' Appeal Brief filed November 28, 2005, as attempting to appeal new claims 41-47 which have not been rejected in any previous Office action. Prior to submitting the Appeal brief, Appellants submitted an amendment to add new claims 41-47 and explained that the new claims 41-47 were based entirely on the previously canceled claims 26-32 which had been finally rejected by the Examiner. Claims 41-47 did not raise new issues that would require further consideration and/or search since the Examiner had finally rejected claims 26-32. Appellants' Appeal brief was submitted assuming the Examiner would enter new claims 41-47 for the reasons provided above, but the Examiner refused to enter the amendments. Merely to expedite prosecution of the remaining claims, Appellants submit a new Appeal brief in compliance with the Examiner's refusal to enter the amendments filed on November 17, 2005.

Respectfully submitted,

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
Legal Dept., M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

Alan D. Christenson, Reg. No. 54,036
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
AGENT FOR APPELLANTS

PAGE 1/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

RECEIVED
CENTRAL FAX CENTER

MAR 14 2006

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellants:	Steven O. HOBBS et al.	§	Confirmation No.:	4992
Serial No.:	09/785,143	§	Group Art Unit:	2192
Filed:	02/16/2001	§	Examiner:	Andre R. Fowlkes
For:	Method And Apparatus For Reducing Cache Thrashing	§	Docket No.:	200301732-1

I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571) 273-8300.
Date of facsimile: 03/14/2006
Typed Name: Colleen F. Brown
Signature: Colleen F. Brown

APPEAL BRIEF

Mail Stop Appeal Brief – Patents
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Date: March 14, 2006

Sir:

Appellants hereby submit this Appeal Brief in connection with the above-identified application. A Notice of Appeal was filed via facsimile on September 29, 2005.

PAGE 2/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST 3
II. RELATED APPEALS AND INTERFERENCES 4
III. STATUS OF THE CLAIMS 5
IV. STATUS OF THE AMENDMENTS 6
V. SUMMARY OF THE CLAIMED SUBJECT MATTER 7
VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL 17
VII. ARGUMENT 18
A. Overview of Carr 18
B. Overview of McGehearty 19
C. Did amendments to claims 1, 9, 17, 24, 25, 33 and 35 introduce new matter? 20
D. Claims 1-8, with claim 1 representing this group 20
E. Claims 9-16, with claim 9 representing this group 22
F. Claims 17-23, with claim 17 representing this group 23
G. Claims 24, 35-37 and 39-40, with claim 35 representing this group 24
H. Claims 33 and 34, with claim 33 representing this group 25
VIII. CONCLUSION 26
IX. CLAIMS APPENDIX 27
X. EVIDENCE APPENDIX 36
XI. RELATED PROCEEDINGS APPENDIX 37

PAGE 3/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

I. REAL PARTY IN INTEREST

The real party in interest is the Hewlett-Packard Development Company (HPDC), a Texas Limited Partnership, having its principal place of business in Houston, Texas. HPDC is a wholly owned affiliate of Hewlett-Packard Company (HPC). HPC merged with Compaq Computer Corporation (CCC) which owned Compaq Information Technologies Group, L.P. (CITG). The Assignment from the inventors to CCC was recorded on February 16, 2001, at Reel/Frame 011614/0531. The Assignment from CCC to CITG was recorded on June 7, 2005, at Reel/Frame 016306/0921. The Change of Name document was recorded on June 8, 2005, at Reel/Frame 016313/0854.

PAGE 4/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals or interferences.

PAGE 5/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

III. STATUS OF THE CLAIMS

In the final Office action dated July 27, 2005, the Examiner rejected claims 1-40. Appellants submitted a response to the final Office action on September 27, 2005, in which Appellants unintentionally canceled claims 26-32. On November 17, 2005, Appellants submitted an Amendment after Notice of Appeal to cancel claims 25 and 38 and to insert the content of the previously canceled claims 26-32 into new claims 41-47. The Examiner entered the amendment canceling claims 26-32, but refused to enter the amendment adding claims 41-47 and canceling claims 25 and 38. Appellants are appealing the rejection of claims 1-25 and 33-40.

PAGE 6/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

IV. STATUS OF THE AMENDMENTS

A Response to Final Office Action dated July 27, 2005, was filed with the U.S. Patent and Trademark Office on September 27, 2005. In the Response, claims 26-32 were unintentionally canceled. On November 17, 2005, Appellants submitted an Amendment after Notice of Appeal to cancel claims 25 and 38 and to insert the content of the previously canceled claims 26-32 into new claims 41-47. The amendment was not a substantive amendment, does not require further searching, and merely puts the remaining claims in better condition for appeal. The Examiner entered the amendment canceling claims 26-32, but refused to enter the amendment adding claims 41-47 and canceling claims 25 and 38. The Claims Appendix includes a copy of the claims to be considered for appeal (claims 1-25 and 33-40).

PAGE 7/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The following provides a concise explanation of the subject matter defined in each of the claims involved in the appeal, referring to the specification by page and line number and to the drawings by reference characters, as required by 37 C.F.R. § 41.37(c)(1)(v). Each element of the claims is identified by a corresponding reference to the specification and drawings where applicable (*i.e.,* the claims are annotated using parenthesis). Note that the citation to passages in the specification and drawings for each claim element does not imply that the limitations from the specification and drawings should be read into the corresponding claim element.

1. (Previously presented) A method, comprising:

identifying a loop in a program (see Fig. 4 and p. 13, lines 23-25);

identifying each vector memory reference in the loop (see Fig. 4; p. 13, lines 5-7; and p. 13, lines 16-19);

determining dependencies between vector memory references in the loop, including determining unidirectional and circular dependencies (see Fig. 4 and p. 13, line 19 – p. 14, line 2);

distributing the vector memory references into a plurality of detail loops configured to allocate the vector memory references into a plurality of temporary arrays, sized and located, so that none of the vector memory references are cache synonyms, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop, and wherein the detail loops are ordered according to the unidirectional dependencies between the memory references (see Fig 4 and p. 14, line 21 – p. 15, line 24);

analyzing an execution profile of the program after said distributing (see Fig. 2 and p. 8, line 11 – p. 9, line 7); and

based on the execution profile, determining whether to repeat said identifying a loop, said identifying each vector memory reference,

PAGE 8/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

said determining dependencies, and said distributing (see Fig. 2 and p. 8, line 11 – p. 9, line 7).

2. (Original) A method, as set forth in claim 1, further comprising allocating a plurality of temporary storage areas within a cache and determining the size of each temporary storage area based on the size of the cache and the number of temporary storage areas (p. 15, lines 12-24).

3. (Original) A method, as set forth in claim 1, further comprising at least one section loop including the plurality of detail loops (p. 15, lines 4-5).

4. (Original) A method, as set forth in claim 1, wherein distributing the vector memory references into a plurality of detail loops further comprises distributing the vector memory references into a plurality of detail loops that each contain at least one vector memory reference that could benefit from cache management (see 320 of Fig. 4 and p. 14, lines 4-14).

5. (Original) A method, as set forth in claim 1, further comprising inserting cache management instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, lines 6-9 and p. 20, lines 27-37).

6. (Original) A method, as set forth in claim 1, further comprising inserting prefetch instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, lines 9-25).

7. (Original) A method, as set forth in claim 1, further comprising performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 19, line 1 – p. 20, line 37).

PAGE 9/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

8. (Original) A method, as set forth in claim 1, further comprising inserting at least one of a prefetch instruction and a cache management instruction into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory, and performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, line 7 – p. 20, line 37).

9. (Previously presented) A method, comprising:

identifying a loop in a program (see Fig. 4 and p. 13, lines 23-25);

identifying each vector memory reference in the loop (see Fig. 4; p. 13, lines 5-7; and p. 13, lines 16-19);

determining dependencies between vector memory references in the loop (see Fig. 4 and p. 13, line 19 – p. 14, line 2); and

distributing the vector memory references into a plurality of detail loops that serially proceed through strips of the vector memory references and store the strips in temporary arrays so that none of the vector memory references are cache synonyms, wherein the vector memory references that have dependencies therebetween are included in a common detail loop (see Fig 4 and p. 14, line 21 – p. 15, line 24);

wherein said distributing the vector memory references into a plurality of detail loops is performed by a first computer for execution by a second computer (p. 7, lines 19-22).

10. (Original) A method, as set forth in claim 9, further comprising allocating a plurality of temporary storage areas within a cache and determining the size of each temporary storage area based on the size of the cache and the number of temporary storage areas (p. 15, lines 12-24).

PAGE 10/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

11. (Original) A method, as set forth in claim 9, further comprising at least one section loop including the plurality of detail loops (p. 15, lines 4-5).

12. (Original) A method, as set forth in claim 9, wherein distributing the vector memory references into a plurality of detail loops further comprises distributing the vector memory references into a plurality of detail loops that each contain at least one vector memory reference that could benefit from cache management (see 320 of Fig. 4 and p. 14, lines 4-14).

13. (Original) A method, as set forth in claim 9, further comprising inserting cache management instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, lines 6-9 and p. 20, lines 27-37).

14. (Original) A method, as set forth in claim 9, further comprising inserting prefetch instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, lines 9-25).

15. (Original) A method, as set forth in claim 9, further comprising performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 19, line 1 – p. 20, line 37).

16. (Original) A method, as set forth in claim 9, further comprising inserting at least one of a prefetch instruction and a cache management instruction into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory, and performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, line 7 – p. 20, line 37).

PAGE 11/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

17. (Previously presented) A method, comprising:

identifying a loop in a program (see Fig. 4 and p. 13, lines 23-25);

identifying each vector memory reference in the loop (see Fig. 4; p. 13, lines 5-7; and p. 13, lines 16-19);

determining dependencies between vector memory references in the loop (see Fig. 4 and p. 13, line 19 – p. 14, line 2);

distributing the vector memory references into a plurality of detail loops in response to cache behavior and the dependencies between the vector memory references in the loop, wherein the detail loops cause storage of the vector memory references in temporary arrays that are allocated consecutively so that no temporary arrays elements are cache synonyms (see Fig 4 and p. 14, line 21 – p. 15, line 24),

wherein said identifying a loop, said identifying each vector memory reference, said determining dependencies between vector memory references and said distributing the vector memory references into a plurality of detail loops produce code that is substantially independent of a computer architecture (p. 8, lines 1-7); and

performing code optimizations that are dependent on a computer architecture after said distributing (p. 8, lines 18-22).

18. (Original) A method, as set forth in claim 17, wherein distributing the vector memory references further comprises distributing the vector memory references into the plurality of detail loops with each loop having at least one of the identified vector memory references (see Fig 4 and p. 14, line 21 – p. 15, line 24).

19. (Original) A method, as set forth in claim 17, further comprising determining dependencies between vector memory references in the loop, and wherein distributing the loop includes distributing the vector memory references into the plurality of detail loops, wherein the vector memory references that have

PAGE 12/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

circular dependencies therebetween are included in a common detail loop (p. 13, lines 16-21).

20. (Original) A method, as set forth in claim 17, further comprising inserting cache management instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, lines 6-9 and p. 20, lines 27-37).

21. (Original) A method, as set forth in claim 17, further comprising inserting prefetch instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, lines 9-25).

22. (Original) A method, as set forth in claim 17, further comprising performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 19, line 1 – p. 20, line 37).

23. (Original) A method, as set forth in claim 17, further comprising inserting at least one of a prefetch instruction and a cache management instruction into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory, and performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory (p. 18, line 7 – p. 20, line 37).

24. (Previously presented) A computer programmed to perform a method, comprising:

identifying a loop in a program (see Fig. 4 and p. 13, lines 23-25);

identifying each vector memory reference in the loop (see Fig. 4; p. 13, lines 5-7; and p. 13, lines 16-19);

PAGE 13/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

determining dependencies between vector memory references in the loop (see Fig. 4 and p. 13, line 19 – p. 14, line 2); and

distributing the vector memory references into a plurality of detail loops configured to retrieve strips of the vector memory references and store the strips in temporary arrays, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop (see Fig 4 and p. 14, line 21 – p. 15, line 24),

wherein the temporary arrays are configured to simultaneously fit in a single cache bank (p. 15, lines 16-20 and p. 17, line 11-15).

25. (Previously presented) A program storage medium encoded with instructions that, when executed by a computer, perform a method, comprising:

identifying a loop in a program (see Fig. 4 and p. 13, lines 23-25);

identifying each vector memory reference in the loop (see Fig. 4; p. 13, lines 5-7; and p. 13, lines 16-19);

determining dependencies between vector memory references in the loop see (Fig. 4 and p. 13, line 19 – p. 14, line 2); and

generating an expanded code of the program by distributing the vector memory references into a plurality of detail loops configured to allocate the vector memory references into temporary arrays that avoid cache synonyms, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop (see Fig. 4 page 8, lines 1-5, and p. 14, line 21 – p. 15, line 24),

wherein the expanded code is substantially independent of computer architectures (p. 8, lines 1-7).

33. (Previously presented) A method for reducing the likelihood of cache thrashing by software to be executed on a computer system (100) having a cache (210, 220), comprising:

PAGE 14/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

executing the software on the computer system (100) (Fig. 1 and p. 7, lines 6-13);

generating a profile indicating the manner in which the software uses the cache (210, 220) (Fig. 2 and p. 8, lines 11-22);

identifying a portion of the software that exhibits cache thrashing based on the profile data (p. 8, lines 11-22 and p. 12, line 23 – p. 13, line 3); and

modifying the identified portion of the software to reduce the likelihood of cache thrashing by distributing cache synonyms into detail loops configured to allocate the cache synonyms into temporary storage areas, sized and located, to prevent cache thrashing (p. 14, line 21– p. 15, line 24),

wherein said modifying occurs before optimizations that are based on an architecture of the computer system (100) (p. 8, lines 14-22).

34. (Previously presented) A method, as set forth in claim 33, wherein modifying the identified portion of the software to reduce the likelihood of cache thrashing further comprises:

identifying a loop in the identified portion of the software (see Fig. 4 and p. 13, lines 23-25);

identifying each vector memory reference in the identified loop (see Fig. 4; p. 13, lines 5-7; and p. 13, lines 16-19);

determining dependencies between the vector memory references in the identified loop of the software, including determining unidirectional and circular dependencies (see Fig. 4 and p. 13, line 19 – p. 14, line 2); and

reducing cache thrashing by distributing the vector memory references into a plurality of detail loops, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop, and wherein the detail loops are ordered

PAGE 15/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

according to the unidirectional dependencies between the memory references (see Fig 4 and p. 14, line 21 – p. 15, line 24).

35. (Previously presented) A method for reducing the likelihood of cache thrashing by software to be executed on a computer system (100) having a cache (210, 220), comprising:

executing the software on the computer system (100) (Fig. 1 and p. 7, lines 6-13);

generating a profile indicating the manner in which the memory references of the software use the cache (210, 220) (Fig. 2 and p. 8, lines 11-22);

identifying a portion of the memory references based on the profile, wherein the portion of the memory references is determined to cause cache thrashing (p. 8, lines 11-22 and p. 12, line 23 – p. 13, line 3); and

reducing cache thrashing by distributing the portion of the memory references into distinct loops that allocate strips of the memory references into temporary arrays for execution (p. 14, line 21 – p. 15, line 24),

wherein the temporary arrays are configured to simultaneously fit in a single cache bank (p. 15, lines 16-20 and p. 17, line 11-15).

36. (Previously presented) The computer of claim 24 wherein the temporary arrays are allocated consecutively such that no temporary array elements are cache synonyms (p. 17, lines 10-15).

37. (Previously presented) The computer of claim 24 wherein the details loops are allocated into section loops that cause iterative execution of the detail loops based on a size of the strips (p. 17, lines 17-27).

PAGE 16/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

38. (Previously presented) The program storage medium of claim 25 wherein the temporary arrays are allocated consecutively such that no temporary array elements are cache synonyms.

39. (Previously presented) The method of claim 35 wherein the temporary arrays are located and sized to reduce cache thrashing (p. 15, lines 12-24).

40. (Previously presented) The method of claim 35 wherein the temporary arrays are allocated consecutively and iteratively executed by a size of the strips (p. 17, lines 10-27).

PAGE 17/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Did amendments to claims 1, 9, 17, 24, 25, 33 and 35 introduce new matter?

Whether claims 1-8 are unpatentable over "Compiler Optimizations for Improving Data Locality" by Steve Carr et at. (hereinafter referred to as the "Carr" reference) in view of U.S. Pat. No. 6,029,225 ("McGehearty").

Whether claims 9-16 are unpatentable over Carr in view of McGehearty.

Whether claims 17-23 are unpatentable over Carr in view of McGehearty.

Whether claims 24, 35-37 and 39-40 are unpatentable over Carr in view of McGehearty.

Whether claims 33-34 are unpatentable over Carr in view of McGehearty.

PAGE 18/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

VII. ARGUMENT

The claims do not stand or fall together. Instead, Appellants present separate arguments for various independent and dependent claims. Each of these arguments is separately argued below and presented with separate headings and sub-headings as required by 37 C.F.R. § 41.37 (c)(1)(vii).

A. Overview of Carr

Carr teaches a compiler that optimizes data locality based on a model that computes both temporal and spatial reuse of cache lines to find desirable loop organizations. In describing the compiler, Carr states:

> The cost model drives the application of compound transformations consisting of loop transformation, loop fusion, loop distribution, and loop reversal.

Page 252, Col. L, lines 7-10

> By driving heuristics with a cache model, our algorithms are efficient and usually find the best transformations for data locality using permutation, fusion and distribution.

Page 253, Col. L, lines 47-50

> Loop distribution separates independent statements in a single loop into multiple loops with identical headers. ...In our system we only use loop distribution to indirectly improve reuse by enabling loop permutation on a nest that is not permutable. ...We only call *Distribute* if memory order cannot be achieved on a nest and not all the inner nests can be fused. ...We thus perform distribution only if it combines with permutation to improve the actual LoopCost.

Page 256, Col. L, line 48 – Col. R, line 33

> Distribution could also be effective if there is no temporal locality between partitions and the accessed arrays are too numerous to fit in cache at once, or register pressure is a concern. We do not address these issues here.

Page 256, Footnote 3

Based on the above, Carr suggests that distribution could be effective if accessed arrays are too numerous to fit in cache at once, but states "we do not address these concerns here." Accordingly, while Carr teaches a compiler that uses loop distribution in some circumstances, Carr does not address using distribution when accessed arrays are too numerous to fit in cache at once. In

PAGE 19/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

contrast, Appellants' claimed invention uses distribution when accessed arrays are too numerous to fit in cache at once (causing cache thrashing).

B. Overview of McGehearty

McGehearty teaches a cache collision avoidance mechanism. In describing the mechanism, McGehearty states:

> [i]f the source and destination addresses have different cache addresses then the data flow into and out of the cache operates in an enhanced mode. If the source and destination addresses have nearly matching cache addresses then the data flow into and out of the cache operates in a near cache collision mode. If the source and destination addresses have exactly matching addresses then the data flow into and out of the cache operates in an exact cache collision avoidance mode.

Col. 2, lines 19-27

> The enhanced mode of cache bank conflict avoidance carefully orders loads and stores so that loads to one memory bank are performed on the same clock cycles as the stores to the other memory bank. After a group of loads and stores are completed, then operations for each bank are switched so that a group of stores are performed on one bank and a group of loads are performed to the other bank.

Col. 2, lines 27-39

> The near cache collision avoidance mode determines the relative locations of the source and destination addresses within the cache. If the source address is slightly after the destination, then the enhanced mode will function properly. If the destination address is slightly after the source address, the groups of cache lines is loaded into registers, and then the registers are stored to memory without any interleaving of other loads and stores.

Col. 2, lines 40-47

> The exact cache collision avoidance mode restructures the loop of moving data to form a series of loads to get several cache lines staged for loading, each element of data is not only moved into cache, but into registers. This is pipelined so that after this initial set of loads is performed, then additional loads are interleaved with non-cache conflicting stores to move new values into memory. By separating the time of loads and stores for matching cache lines, full pipelining and multi-entry cache benefits is obtained.

Col. 2, lines 48-57

PAGE 20/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

While McGehearty does teach a mechanism for cache collision avoidance, the mechanism is not based on "detail loops" as required in Appellants' claimed invention. Instead, McGehearty implements a cache manager 19 that monitors and controls the flow of data into and out of the cache 10. When data is being loaded into the cache 10 from memory 15, the cache manager 19 compares (by subtraction) the memory source address and the memory destination address to determine whether to implement the enhanced mode, the near cache collision mode or the exact cache collision mode. In contrast, Appellants' claimed invention reduces cache thrashing (collisions) by distributing vector memory references into detail loops.

C. Did amendments to claims 1, 9, 17, 24, 25, 33 and 35 introduce new matter?

As shown in the "SUMMARY OF THE CLAIMED SUBJECT MATTER" section, the limitations of claims 1, 9, 17, 24, 25, 33 and 35 are supported by Appellants' specification. Accordingly, the Examiner improperly objected to amendments to claims 1, 9, 17, 24, 25, 33 and 35 as introducing new matter under 35 U.S.C § 132 and 37 C.F.R. § 1.121(f).

D. Claims 1-8, with claim 1 representing this group

Claim 1, in part, requires "determining dependencies between vector memory references in [a] loop, including determining unidirectional and circular dependencies." Claim 1 further requires "distributing the vector memory references into a plurality of detail loops configured to allocate the vector memory references into a plurality of temporary arrays, sized and located, so that none of the vector memory references are cache synonyms." Claim 1 further requires "analyzing an execution profile of the program after said distributing" and "based on the execution profile, determining whether to repeat...said determining dependencies, and said distributing."

The Examiner's obviousness rejection of claim 1 in view of Carr and McGehearty is improper for at least three reasons. First, the Examiner admits that Carr does not disclose Appellants' claimed "allocat[ing] the vector memory references into a plurality of temporary arrays, sized and located, so that none of

PAGE 21/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

the vector memory references are cache synonyms," and relies on McGehearty as teaching this limitation (see final Office action, page 4, third paragraph). However, McGehearty does not appear to teach or suggest "detail loops" are used to "allocate the vector memory references into a plurality of temporary arrays" as required in claim 1. Instead, McGehearty teaches a cache manager 19 that monitors and controls the flow of data into and out of the cache 11. "To avoid cache collisions that occur when there is an exact mapping of source and destination addresses, the manager 19 loads the data from the memory that would fill multiple cache lines into different registers, via the cache, to start a pipelined copy operation" (See Fig. 1 and Col. 6, lines 23-27). Appellants cannot find a teaching or suggestion in McGehearty that "detail loops...allocate the vector memory references into a plurality of temporary arrays" as required in claim 1.

Second, there is no clear motivation to combine Carr with McGehearty "to avoid cache collisions" as suggested by the Examiner. As previously discussed, Carr optimizes data locality based on a cost model. "The model computes both temporal and spatial reuse of cache lines to find desirable loop organizations" (Page 252, Col. L, lines 7-8). In contrast, McGehearty's cache collision avoidance mode restructures loops when a cache manager 19 determines that data to be stored in a cache 11 will cause cache collisions (Col. 3, lines 50-55 and col. 6, lines 23-32). Because Carr's algorithm is intended to optimize data locality and loop organizations, one of ordinary skill in the art would not think to combine with Carr with McGehearty's loop restructuring mechanism. In other words, McGehearty's loop restructuring mechanism would not be needed if loops have been organized to optimize data locality according to Carr. Likewise, if McGehearty's loop restructuring mechanism is implemented, there is no apparent need to optimize data locality and loop organization according to Carr.

Third, none of the references cited by the Examiner, nor combinations of the references, teaches or suggests "analyzing an execution profile of the program after said distributing" and "based on the execution profile, determining whether to repeat...said determining dependencies, and said distributing" as

PAGE 22/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

required in claim 1. On the contrary, Carr states "our approach only performs one evaluation step because it directly determines the best loop permutation" (Page 253, Col. L, lines 22-24). For at least these reasons, Appellants submit that the Examiner improperly rejected claim 1 and that claims 1-8 should be allowed.

E. Claims 9-16, with claim 9 representing this group

Claim 9, in part, requires "distributing the vector memory references into a plurality of detail loops that serially proceed through strips of the vector memory references and store the strips in temporary arrays so that none of the vector memory references are cache synonyms." Claim 9 further requires "said distributing the vector memory references into a plurality of detail loops is performed by a first computer for execution by a second computer."

The Examiner's obviousness rejection of claim 9 in view of Carr and McGehearty is improper for at least three reasons. First, the Examiner admits that Carr does not disclose Appellants' claimed "a plurality of detail loops that serially proceed through strips of vector memory references and store the strips in temporary arrays so that none of the vector memory references are cache synonyms" and relies on McGehearty as teaching this limitation (see Final Office Action, page 8, first and second paragraphs). However, McGehearty does not appear to teach or suggest "detail loops that serially proceed through strips of vector memory references and store the strips in temporary arrays so that none of the vector memory references are cache synonyms" as required in claim 9. Instead, McGehearty teaches "[t]o avoid cache collisions that occur when there is an exact mapping of source and destination addresses, the [cache] manager 19 loads the data from the memory that would fill multiple cache lines into different registers, via the cache, to start a pipelined copy operation" (See Fig. 1 and Col. 6, lines 23-27). Thus, McGehearty appears to teach a cache manager 19 rather than Appellants' claimed "detail loops" is responsible for cache collision avoidance.

Second, as previously discussed with respect to claim 1, there is no clear motivation to combine Carr with McGehearty "to avoid cache collisions" as suggested by the Examiner. Third, none of the references cited by the Examiner,

PAGE 23/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

nor combinations of the references, appear to teach or suggest "said distributing the vector memory references into a plurality of detail loops is performed by a first computer for execution by a second computer" as required in claim 9. For at least these reasons, Appellants submit that the Examiner improperly rejected claim 9 and that claims 9-16 should be allowed.

F. Claims 17-23, with claim 17 representing this group

Claim 17, in part, requires "distributing the vector memory references into a plurality of detail loops in response to cache behavior and the dependencies between the vector memory references in the loop, wherein the detail loops cause storage of the vector memory references in temporary arrays that are allocated consecutively so that no temporary [array] elements are cache synonyms." Claim 17 further requires "said distributing the vector memory references into a plurality of detail loops produce[s] code that is substantially independent of a computer architecture" and "performing code optimizations that are dependent on a computer architecture after said distributing."

The Examiner's obviousness rejection of claim 17 in view of Carr and McGehearty is improper for at least three reasons. First, the Examiner recognizes that Carr does not teach Appellants' claimed "storage of the vector memory references in temporary arrays that are allocated consecutively so that no temporary [array] elements are cache synonyms" and relies on McGehearty as teaching this limitation (see final Office action, page 10, first and second paragraphs). However, McGehearty does not appear to teach or suggest "detail loops [that] cause storage of the vector memory references in temporary arrays that are allocated consecutively so that no temporary [array] elements are cache synonyms" as required in claim 17. Instead of Appellants' claimed "detail loops," McGehearty relies on a cache manager 19 that loads data from the memory that would fill multiple cache lines into different registers, via the cache, to start a pipelined copy operation.

Second, as previously discussed with respect to claim 1, there is no clear motivation to combine Carr with McGehearty "to avoid cache collisions" as suggested by the Examiner. Third, none of the references cited by the Examiner,

PAGE 24/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

nor combinations of the references, appear to teach or suggest "said distributing the vector memory references into a plurality of detail loops produce[s] code that is substantially independent of a computer architecture" and "performing code optimizations that are dependent on a computer architecture after said distributing." For at least these reasons, Appellants submit that the Examiner improperly rejected claim 17 and that claims 17-23 should be allowed.

G. Claims 24, 35-37 and 39-40, with claim 35 representing this group

Claim 35, in part, requires "identifying a portion of the memory references based on the profile, wherein the portion of the memory references is determined to cause cache thrashing." Claim 35 further requires "reducing cache thrashing by distributing the portion of the memory references into distinct loops that allocate strips of the memory references into temporary arrays for execution." Claim 35 further required "the temporary arrays are configured to simultaneously fit in a single cache bank."

The Examiner's obviousness rejection of claim 35 in view of Carr and McGehearty is improper for at least three reasons. First, none of the references cited by the Examiner, nor combinations of the references, teaches or suggests "identifying a portion of the memory references...determined to cause cache thrashing" and "reducing cache thrashing by distributing the portion of the memory references into distinct loops that allocate strips of the memory references into temporary arrays for execution" as required in claim 35. While McGehearty teaches a cache collision avoidance mode that "restructures [a] loop of moving data," McGehearty does not teach or suggest Appellants' claimed "distributing the portion of the memory references into distinct loops that allocate strips of the memory references into temporary arrays for execution."

Second, as previously discussed with respect to claim 1, there is no clear motivation to combine Carr with McGehearty "to avoid cache collisions" as suggested by the Examiner. Third, none of the references cited by the Examiner, nor combinations of the references, appear to teach or suggest "the temporary arrays are configured to simultaneously fit in a single cache bank" as required in

PAGE 25/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

claim 35. The Examiner cites Carr as teaching "knowledge of the cache size, associativity, and replacement policy is essential" (see final Office action, page 5, last paragraph). However, the Carr reference still does not teach or suggest "temporary arrays are configured to simultaneously fit in a single cache bank" as required in claim 35. McGehearty is likewise deficient in this regard. For at least these reasons, Appellants submit that the Examiner improperly rejected claim 35 and that claims 24, 35-37 and 39-40 should be allowed.

H. Claims 33 and 34, with claim 33 representing this group

Claim 33, in part, requires "modifying the identified portion of the software to reduce the likelihood of cache thrashing by distributing cache synonyms into detail loops configured to allocate the cache synonyms into temporary storage areas, sized and located, to prevent cache thrashing." Claim 33 further requires "said modifying occurs before optimizations that are based on an architecture of the computer system."

The Examiner's obviousness rejection of claim 33 in view of Carr and McGehearty is improper for at least three reasons. First, the Examiner admits that Carr does not teach "distributing cache synonyms into detail loops configured to allocate the cache synonyms into temporary storage areas, sized and located, to prevent cache thrashing" and relies on McGehearty as teaching this limitation. However, McGehearty does not appear to teach or suggest "detail loops configured to allocate the cache synonyms into temporary storage areas, sized and located, to prevent cache thrashing" as required in claim 33. Instead of Appellants' claimed "detail loops," McGehearty relies on a cache manager 19 that loads data from the memory that would fill multiple cache lines into different registers, via the cache, to start a pipelined copy operation.

Second, as previously discussed with respect to claim 1, there is no clear motivation to combine Carr with McGehearty "to avoid cache collisions" as suggested by the Examiner. Third, none of the references cited by the Examiner, nor combinations of the references, appear to teach or suggest "said [distributing cache synonyms into detail loops] occurs before optimizations that are based on an architecture of the computer system." For at least these reasons, Appellants

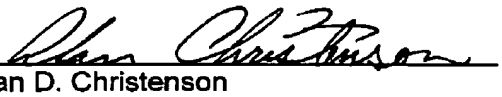
PAGE 26/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

submit that the Examiner improperly rejected claim 33 and that claims 33 and 34 should be allowed.

VIII. CONCLUSION

For the reasons stated above, Appellants respectfully submit that the Examiner erred in rejecting all pending claims. It is believed that no extensions of time or fees are required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Hewlett-Packard Development Company's Deposit Account No. 08-2025.

Respectfully submitted,



Alan D. Christenson
PTO Reg. No. 54,036
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
AGENT FOR APPELLANTS

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
Legal Dept., M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

PAGE 27/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

IX. CLAIMS APPENDIX

1 (Previously presented) A method, comprising:

identifying a loop in a program;

identifying each vector memory reference in the loop;

determining dependencies between vector memory references in the loop, including determining unidirectional and circular dependencies;

distributing the vector memory references into a plurality of detail loops configured to allocate the vector memory references into a plurality of temporary arrays, sized and located, so that none of the vector memory references are cache synonyms, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop, and wherein the detail loops are ordered according to the unidirectional dependencies between the memory references;

analyzing an execution profile of the program after said distributing; and

based on the execution profile, determining whether to repeat said identifying a loop, said identifying each vector memory reference, said determining dependencies, and said distributing.

2. (Original) A method, as set forth in claim 1, further comprising allocating a plurality of temporary storage areas within a cache and determining the size of each temporary storage area based on the size of the cache and the number of temporary storage areas.

3. (Original) A method, as set forth in claim 1, further comprising at least one section loop including the plurality of detail loops.

4. (Original) A method, as set forth in claim 1, wherein distributing the vector memory references into a plurality of detail loops further comprises distributing the vector memory references into a plurality of detail loops that each contain at least one vector memory reference that could benefit from cache management.

PAGE 28/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

5. (Original) A method, as set forth in claim 1, further comprising inserting cache management instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

6. (Original) A method, as set forth in claim 1, further comprising inserting prefetch instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

7. (Original) A method, as set forth in claim 1, further comprising performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

8. (Original) A method, as set forth in claim 1, further comprising inserting at least one of a prefetch instruction and a cache management instruction into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory, and performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

9. (Previously presented) A method, comprising:
identifying a loop in a program;
identifying each vector memory reference in the loop;
determining dependencies between vector memory references in the loop; and
distributing the vector memory references into a plurality of detail loops that serially proceed through strips of the vector memory references and store the strips in temporary arrays so that none of the vector

PAGE 29/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

memory references are cache synonyms, wherein the vector memory references that have dependencies therebetween are included in a common detail loop;

wherein said distributing the vector memory references into a plurality of detail loops is performed by a first computer for execution by a second computer.

10. (Original) A method, as set forth in claim 9, further comprising allocating a plurality of temporary storage areas within a cache and determining the size of each temporary storage area based on the size of the cache and the number of temporary storage areas.

11. (Original) A method, as set forth in claim 9, further comprising at least one section loop including the plurality of detail loops.

12. (Original) A method, as set forth in claim 9, wherein distributing the vector memory references into a plurality of detail loops further comprises distributing the vector memory references into a plurality of detail loops that each contain at least one vector memory reference that could benefit from cache management.

13. (Original) A method, as set forth in claim 9, further comprising inserting cache management instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

14. (Original) A method, as set forth in claim 9, further comprising inserting prefetch instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

PAGE 30/38 * RCVD AT 3/14/2006 5:22:48 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

15. (Original) A method, as set forth in claim 9, further comprising performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

16. (Original) A method, as set forth in claim 9, further comprising inserting at least one of a prefetch instruction and a cache management instruction into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory, and performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

17. (Previously presented) A method, comprising:

identifying a loop in a program;

identifying each vector memory reference in the loop;

determining dependencies between vector memory references in the loop;

distributing the vector memory references into a plurality of detail loops in response to cache behavior and the dependencies between the vector memory references in the loop, wherein the detail loops cause storage of the vector memory references in temporary arrays that are allocated consecutively so that no temporary arrays elements are cache synonyms,

wherein said identifying a loop, said identifying each vector memory reference, said determining dependencies between vector memory references and said distributing the vector memory references into a plurality of detail loops produce code that is substantially independent of a computer architecture; and

performing code optimizations that are dependent on a computer architecture after said distributing.

PAGE 31/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

18. (Original) A method, as set forth in claim 17, wherein distributing the vector memory references further comprises distributing the vector memory references into the plurality of detail loops with each loop having at least one of the identified vector memory references.

19. (Original) A method, as set forth in claim 17, further comprising determining dependencies between vector memory references in the loop, and wherein distributing the loop includes distributing the vector memory references into the plurality of detail loops, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop.

20. (Original) A method, as set forth in claim 17, further comprising inserting cache management instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

21. (Original) A method, as set forth in claim 17, further comprising inserting prefetch instructions into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

22. (Original) A method, as set forth in claim 17, further comprising performing loop unrolling on at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory.

23. (Original) A method, as set forth in claim 17, further comprising inserting at least one of a prefetch instruction and a cache management instruction into at least one of said detail loops to control movement of data associated with the vector memory reference between a cache and main memory, and performing loop unrolling on at least one of said detail loops to control movement of data

PAGE 32/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

associated with the vector memory reference between a cache and main memory.

24. (Previously presented) A computer programmed to perform a method, comprising:

identifying a loop in a program;

identifying each vector memory reference in the loop;

determining dependencies between vector memory references in the loop; and

distributing the vector memory references into a plurality of detail loops configured to retrieve strips of the vector memory references and store the strips in temporary arrays, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop,

wherein the temporary arrays are configured to simultaneously fit in a single cache bank.

25. (Previously presented) A program storage medium encoded with instructions that, when executed by a computer, perform a method, comprising:

identifying a loop in a program;

identifying each vector memory reference in the loop;

determining dependencies between vector memory references in the loop; and

generating an expanded code of the program by distributing the vector memory references into a plurality of detail loops configured to allocate the vector memory references into temporary arrays that avoid cache synonyms, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop,

wherein the expanded code is substantially independent of computer architectures.

PAGE 33/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

26-32. (Canceled).

33. (Previously presented) A method for reducing the likelihood of cache thrashing by software to be executed on a computer system having a cache, comprising:

executing the software on the computer system;

generating a profile indicating the manner in which the software uses the cache;

identifying a portion of the software that exhibits cache thrashing based on the profile data; and

modifying the identified portion of the software to reduce the likelihood of cache thrashing by distributing cache synonyms into detail loops configured to allocate the cache synonyms into temporary storage areas, sized and located, to prevent cache thrashing,

wherein said modifying occurs before optimizations that are based on an architecture of the computer system.

34. (Previously presented) A method, as set forth in claim 33, wherein modifying the identified portion of the software to reduce the likelihood of cache thrashing further comprises:

identifying a loop in the identified portion of the software;

identifying each vector memory reference in the identified loop;

determining dependencies between the vector memory references in the identified loop of the software, including determining unidirectional and circular dependencies; and

reducing cache thrashing by distributing the vector memory references into a plurality of detail loops, wherein the vector memory references that have circular dependencies therebetween are included in a common detail loop, and wherein the detail loops are ordered according to the unidirectional dependencies between the memory references.

PAGE 34/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

35. (Previously presented) A method for reducing the likelihood of cache thrashing by software to be executed on a computer system having a cache, comprising:

executing the software on the computer system;

generating a profile indicating the manner in which the memory references of the software use the cache;

identifying a portion of the memory references based on the profile, wherein the portion of the memory references is determined to cause cache thrashing; and

reducing cache thrashing by distributing the portion of the memory references into distinct loops that allocate strips of the memory references into temporary arrays for execution,

wherein the temporary arrays are configured to simultaneously fit in a single cache bank.

36. (Previously presented) The computer of claim 24 wherein the temporary arrays are allocated consecutively such that no temporary array elements are cache synonyms.

37. (Previously presented) The computer of claim 24 wherein the details loops are allocated into section loops that cause iterative execution of the detail loops based on a size of the strips.

38. (Previously presented) The program storage medium of claim 25 wherein the temporary arrays are allocated consecutively such that no temporary array elements are cache synonyms.

39. (Previously presented) The method of claim 35 wherein the temporary arrays are located and sized to reduce cache thrashing.

PAGE 35/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

40. (Previously presented) The method of claim 35 wherein the temporary arrays are allocated consecutively and iteratively executed by a size of the strips.

PAGE 38/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

X. EVIDENCE APPENDIX

None.

PAGE 37/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02

XI. RELATED PROCEEDINGS APPENDIX

None.

PAGE 38/38 * RCVD AT 3/14/2006 5:22:46 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-3/5 * DNIS:2738300 * CSID:7132388008 * DURATION (mm-ss):13-02